CERN

# EOS+CTA Architecture and Functionality

Meeting with STFC - discuss CTA potential adoption

CTA Architecture/Functionality details
Commonalities and differences with CASTOR
Current status and plans

Eric Cano on behalf of the CTA team

## What is CTA?

- CTA is a pure tape system
- One central CTA instance provides the tape backend to several EOS instances

# EOS+CTA architecture: big picture

- EOS as the interface to the outside world
  - is the sole interface to the users (XRootd)
  - holds the directory structure, files names and metadata, disk buffer
  - handle locally operations that do not require the tape backend
  - provides an extended XRootd based interface for tape aware operations
  - handles garbage collection
- CTA as the tape backend
  - manages transparently file residence on tape
  - transfers tape files to/from disk cache on request from EOS
  - manages drives, tapes, libraries, file location, queues, and more...

## Typical user operations

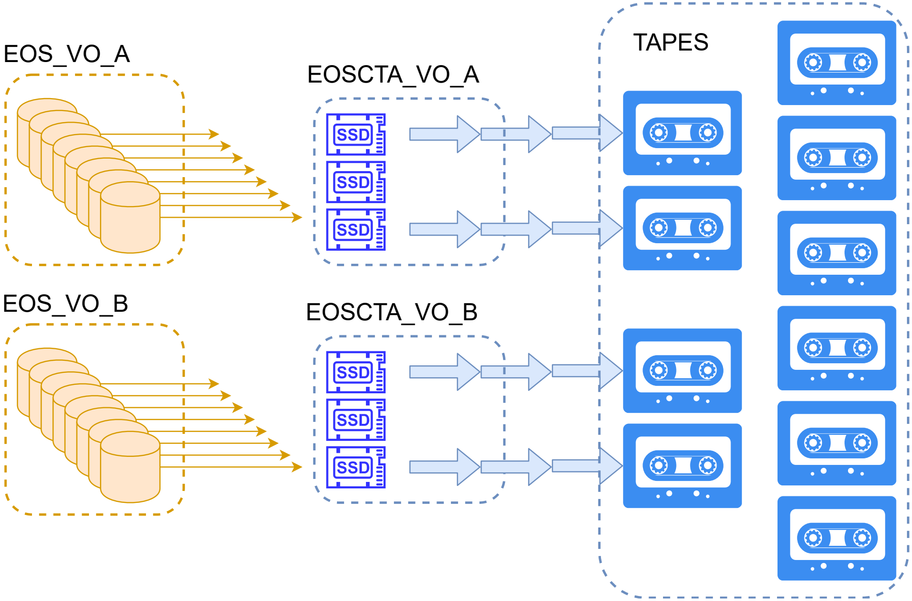
- Write file to EOS+CTA buffer
- Is file on tape?
- Queue file for retrieve
- Is file in EOS+CTA buffer?
- Read file from EOS+CTA buffer
- Evict file from EOS+CTA buffer
- Delete file from EOS+CTA

## Fitting EOS+CTA in the bigger picture

A mode of operations validated with Atlas:

- EOS+CTA used as a tape store only
- SSD based EOS+CTA buffer
- Experiment's EOS instance does the grid transfers
- Multi-step sequence achieved with FTS
- Auto-evict after successful archive
- FTS evicts from buffer after successful retrieve

Other modes of operation possible


EOS_VO_A
EOSCTA_VO_A
TAPES
SSD
SSD
SSD
EOS_VO_B
EOSCTA_VO_B
SSD
SSD
SSD

## A CTA instance in depth

Three daemons:

- cta-taped: Evolution of CASTOR's tapeserverd. An extra subprocess for maintenance.
- cta-frontend: XRootd based server allowing EOS and CLI access to data structures.
- cta-rmcd: also inherited from CASTOR.

Two backend stores:

- Catalogue DB for persistent data.
- Scheduler DB for transient object (requests, queues), implemented as an object store.

A CLI: cta-admin

## Software architecture

3 main software components, shared by taped and frontend:

- Catalogue
- SchedulerDB (also called object store)
- Scheduler links the two.

Both Catalogue and SchedulerDB have several backends:

- Catalogue:
    - Oracle
    - MySQL
    - PostgreSQL
    - SQLite (for unit tests)
- SchedulerDB:
    - Rados
    - VFS (local file system)

## Functionality details (present and planned)

- Tape server: Tape alert support, HW RAO (enterprise drives), SW RAO (LTO)[1], LBP, encryption, empty file skip on write[1].

- Scheduler: Priorities, drive dedication, fair share between VOs[1], intra-VO weighted fair share (activites)[1], mount pre-emption[1], backpressure[1], dataset colocation[1], repack request expansion on demand[1]
- EOS: Garbage collection, retrieve request filtering, retrieve request cancellation, disk copy eviction

[1]Explained in backup slides

## Commonalities with CASTOR: concepts

- Storage class (CTA) $\Leftrightarrow$ File class (CASTOR)
- Archive file (CTA) + Directory entry (EOS) $\approx$ Castorfile
- Libraries, tape pools, tapes
- Migration/archive routes
- Admin users (CTA) $\approx$ CUPV (CASTOR)
- Requester (group) mount rule (CTA) $\approx$ Recall (user) group (CASTOR)

## Differences with CASTOR: architecture (I)

- Namespace handled by EOS
  - Each EOS instance has a disjoint namespace w.r.t. other instances.
  - The shared CTA instance stores tape files for all
  - Tape files simply referenced by numeric archive file Id.
  - Archive file Id stored in EOS namespace as file extended attribute.

## Differences with CASTOR: architecture (II)

- Shared backend storage vs per daemon DBs
  - In CASTOR, DBs are owned by a specific daemon, and accessed from one or few instances.
  - In CTA, each actor accesses directly the backend data, effectively running a collapsed stack of CASTOR daemons in a single process.
  - This was a constraint at the time of the design of CASTOR, not anymore.

## Differences with CASTOR: architecture (III)

- Global decision vs partial decision
  - In CASTOR each actor took decision based on partial information. Action is based on a chain of partial information based decisions, leading to inefficiencies. Typical example: the VMGR decides to write to a tape located in a library where all the drives are busy.
  - In CTA, the decisions are taken at mount time, by the tape server, when it is free. The frontend just updates the shared queues. Drives share their status with each other to allow a fully informed mount decision.

## Differences with CASTOR: architecture (IV)

- Queueing: DB vs object store
  - Queueing in DBs is a hard problem. Some algorithms are $\mathcal{O}(n)$ and executed on each scheduling. Oracle does not handle well tables that grow and shrink. The [redacted] optimizer.
  - Shared object allow $\approx \mathcal{O}(1)$ algorithms. Each queue instance is in a separate object, minimizing contention.

## Differences with CASTOR: Daemons

Far less daemons in CTA:

- Front end: allow manipulation of queues and persistent data by EOS and CLI
- Tape server
  - One process per drive: schedules the mounts for its own drive, does the data transfer and the minimum amount of metadata management.
  - One maintenance process per box: does the backoffice maintenance: repack management, cleanup in case of process crashes.

## Differences with CASTOR: Disk/buffer

Handled in EOS. Will be presented in detail by Julien and Luca.

## Current status and plans

- Already implemented:
    - Tape server: Tape alert support, HW RAO (enterprise drives), LBP, encryption, empty file skip on write
    - Scheduling: Priorities, fair share between VOs, intra-VO weighted fair share (activites), backpressure, repack request expansion on demand
    - EOS: all
- Upcoming features:
    - Tape server: SW RAO (LTO)
    - Scheduling: drive dedication, mount pre-emption, dataset colocation (Pending PhD study)

## Summary

- CTA+EOS provides a system conceptually similar to CASTOR
- It uses newer backends to approach "full speed, full time"
- Preparing the terrain for tape intensive workloads (tape carousel)

CERN

## Backup slides

## New features in detail (I)

- SW RAO: Software recommended access order: store physical location (wrap, position) of files in catalogue and implement a “traveling salesman” algorithm on retrieve.
- Empty file skip: leave a place holder when we discover that the file was deleted by user between queueing and writing to tape. Avoids unmounts.

## New features in detail (II)

- Fair share between VOs: try to give each VO the same proportion of their maximum number of drives (when conditions permit).
- Intra-VO fair share (activities): each file retrieve can be tagged with an activity. Activities are weighted and given a fair share of mounts. Intra VO, and weight based. The feature already exists in FTS and is mirrored in CTA.

## New features in detail (III)

- Mount pre-emption: mount scheduling also happens during the mount (at a low rate) to ensure no higher priority mount could be served. If so, we stop the low priority mount and start the higher priority one. Goal is to run the drives "full speed, full time".

## New features in detail (IV)

- Backpressure: disk space reservation before committing for retrieve, allowing prevention of limited buffer overrun. Prevents error and enables “full speed, full time”.

## New features in detail (V)

- Dataset colocation: allow experiments to tag files with a (possibly hierarchical) dataset. Make sure each dataset is locally contiguous in the tape, to minimize positioning at retrieve time and ensure maximum read performance.

## New features in detail (VI)

- Repack request expansion on demand: repack requests are queued as very small object. A few of them only are blown up into per-file sub requests and actually visible to the tape drives. As the requests get processes, more get expanded. This avoids clogging the queueing system, and to queue an arbitrary number of tapes to be repacked.